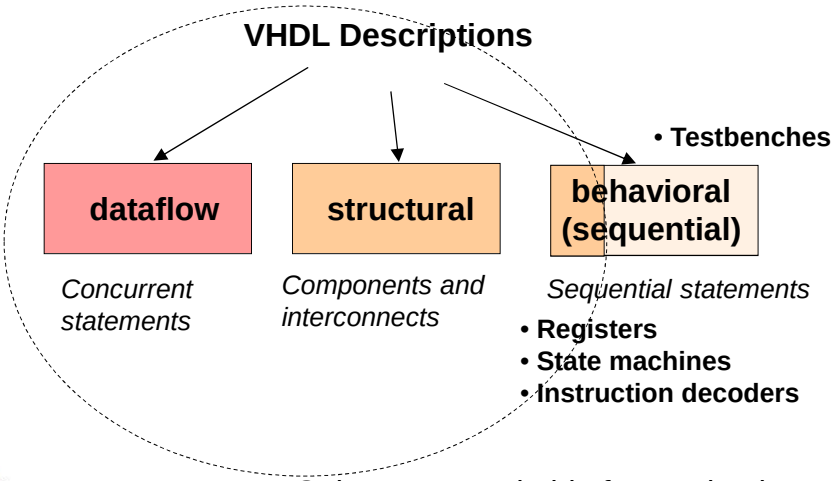
Types of VHDL Description (Modeling Styles)

1

Types of VHDL Description



Subset most suitable for synthesis

2

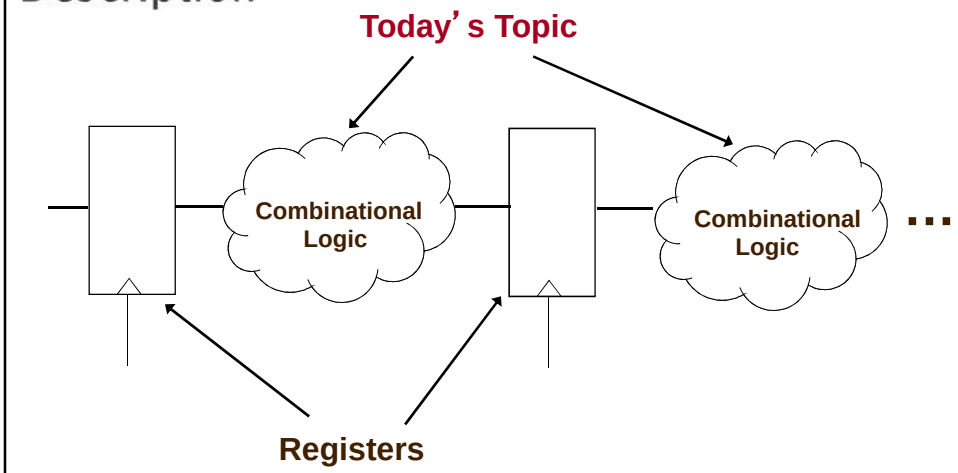
Synthesizable VHDL

Dataflow VHDL Description → VHDL code synthesizable

Dataflow VHDL Description ✗ VHDL code synthesizable

3

Register Transfer Level (RTL) Design Description



4

Data-Flow VHDL

Concurrent Statements

- **simple concurrent signal assignment**
  (Ĕ)
- **conditional concurrent signal assignment**
  (when-else)
- **selected concurrent signal assignment**
  (with-select-when)
- **generate scheme for equations**
  (for-generate)

5

Data-Flow VHDL

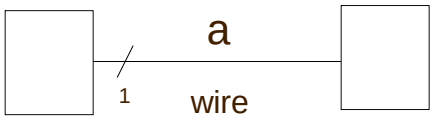
Concurrent Statements

- **simple concurrent signal assignment**
  (Ĕ)
- **conditional concurrent signal assignment**
  (when-else)
- **selected concurrent signal assignment**
  (with-select-when)
- **generate scheme for equations**
  (for-generate)

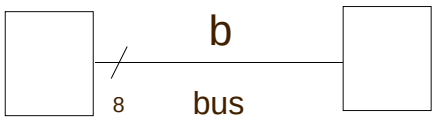
6

Signals

SIGNAL a : STD_LOGIC;

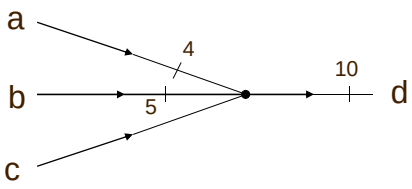


SIGNAL b : STD_LOGIC_VECTOR(7 DOWNTO 0);



7

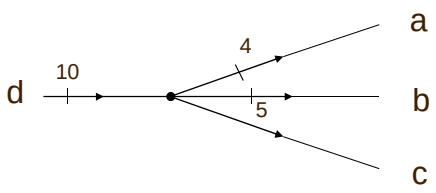
Merging wires and buses




```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(9 DOWNTO 0);

d <= a & b & c;
```

8

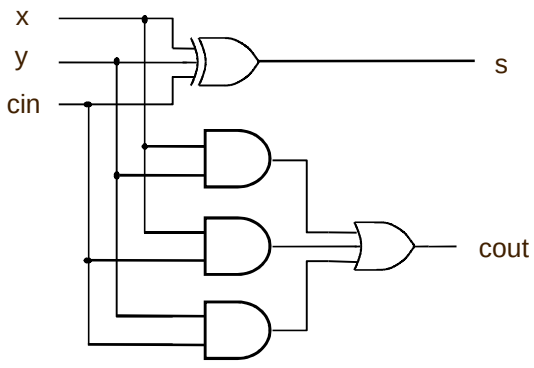
Splitting buses




```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(9 DOWNTO 0);

a <= d(9 downto 6);
b <= d(5 downto 1);
c <= d(0);
```

9

Data-flow VHDL: Example



Data-flow VHDL: Example (1)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY fulladd IS
        PORT ( x    : IN     STD_LOGIC ;
               y    : IN     STD_LOGIC ;
               cin  : IN     STD_LOGIC ;
               s    : OUT    STD_LOGIC ;
               cout: OUT     STD_LOGIC ) ;
END fulladd ;


ARCHITECTURE dataflow OF fulladd IS
BEGIN
        s    <=  x XOR y XOR cin ;
        cout <=  (x AND y) OR (cin AND x) OR (cin AND y) ;
END dataflow ;
```

11

Logic Operators

- Logic operators

```
and   or   nand   nor   xor   not   xnor
```

- Logic operators precedence

only in VHDL-93 or later

Highest ↓ Lowest

```
                  not
and   or   nand   nor   xor   xnor
```

12

No Implied Precedence

Wanted: y = **ab** + **cd**

Incorrect

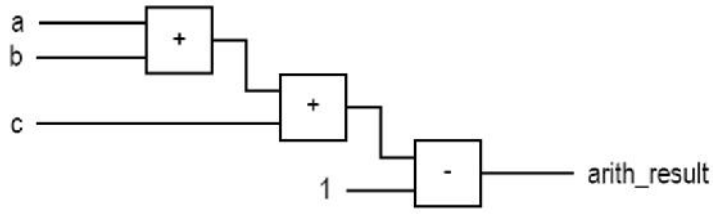
y <= a **and** b **or** c **and** d ;
equivalent to
y <= ((a **and** b) **or** c) **and** d ;
equivalent to
y = (ab + c)d

Correct

y <= (a **and** b) **or** (c **and** d) ;

13

arith_result <= a + b + c - 1;



14

Signal assignment statement with a closed feedback loop

- A signal appears in both sides of a concurrent assignment statement
- E.g.,
  q <= ((**not q**) **and** (**not** en)) **or** (d **and** en);
- Syntactically correct
- Form a closed feedback loop
- Should be avoided

15

Data-Flow VHDL

Concurrent Statements

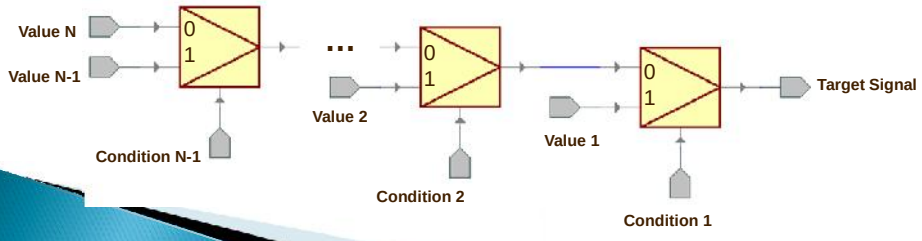
- **simple concurrent signal assignment**
  (Ĕ)
- **conditional concurrent signal assignment**
  (when-else)
- **selected concurrent signal assignment**
  (with-select-when)
- **generate scheme for equations**
  (for-generate)

16

Most often implied structure

When - Else

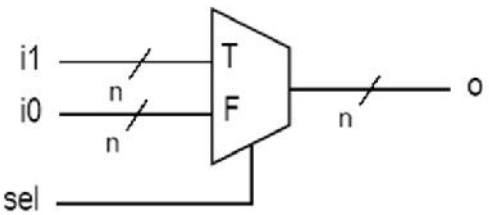
```
target_signal <= value1 when condition1 else
                 value2 when condition2 else
                   . . .
                 valueN-1 when conditionN-1 else
                 valueN;
```



17

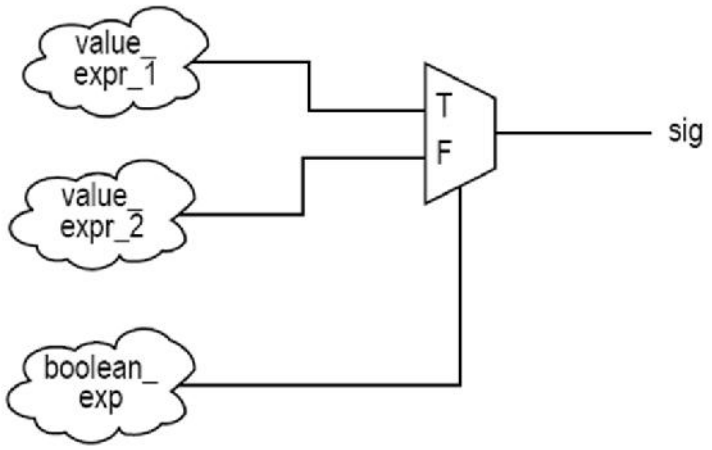
2-to-1 "abstract" mux

- sel has a data type of boolean
- If sel is true, the input from "T" port is connected to output.
- If sel is false, the input from "F" port is connected to output.



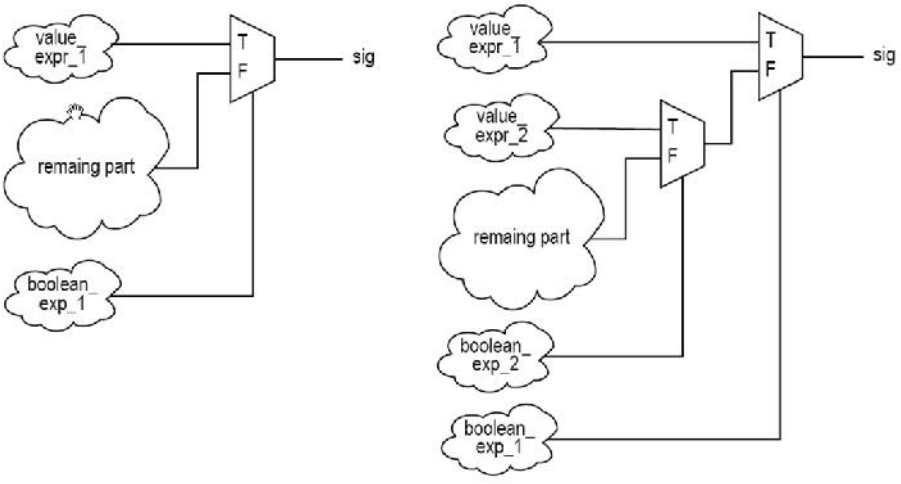
18

```
signal_name <= value_expr_1 when boolean_expr_1 else
               value_expr_2;
```

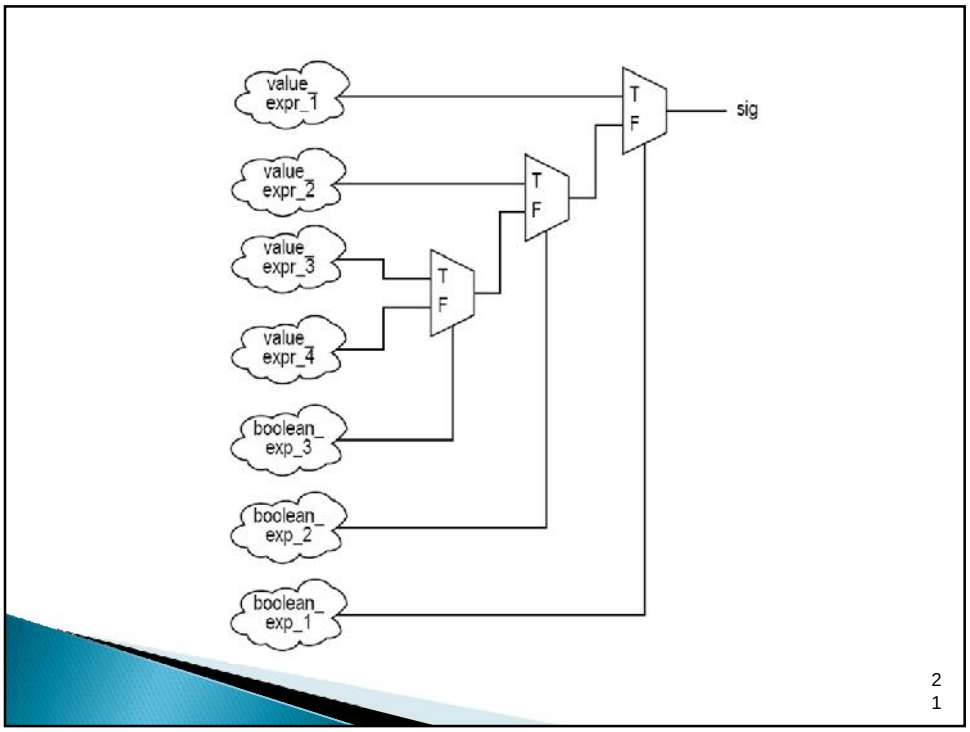


19

```
signal_name <= value_expr_1 when boolean_expr_1 else
               value_expr_2 when boolean_expr_2 else
               value_expr_3 when boolean_expr_3 else
               value_expr_4;
```

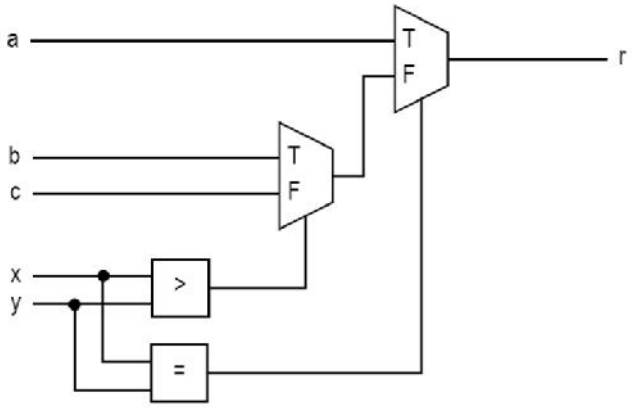


20



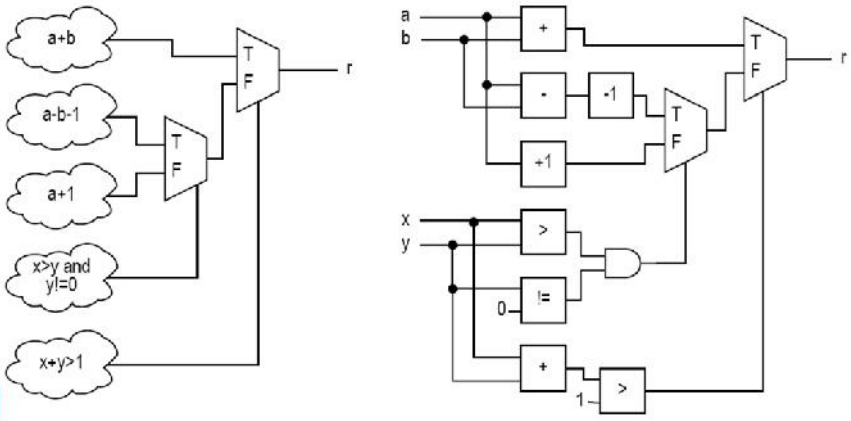
- E.g.,

```
signal a,b,c,x,y,r: std_logic;
. . .
r <= a when x=y else
     b when x>y else
     c;
```



- E.g.,

```
. . .
signal a,b,r: unsigned(7 downto 0);
signal x,y: unsigned(3 downto 0);
. . .
r <= a+b when x+y>1 else
     a-b-1 when x>y and y!=0 else
     a+1;
```



23

Signed and Unsigned Types

Behave exactly **like**

STD_LOGIC_VECTOR

plus, they determine whether a given vector should be treated as a signed or unsigned number.

Require

USE ieee.numeric_std.all;

24

Operators

- Relational operators

```
=     /=     <     <=     >     >=
```

- Logic and relational operators precedence

Highest ↓ Lowest

```
                  not
=     /=     <     <=     >     >=
and   or   nand   nor   xor   xnor
```

25

Priority of logic and relational operators

compare a = bc

Incorrect

… when a = b **and** c else …

equivalent to

… when (a = b) **and** c else …

Correct

… when a = (b **and** c) else …

26

Data-Flow VHDL

Concurrent Statements

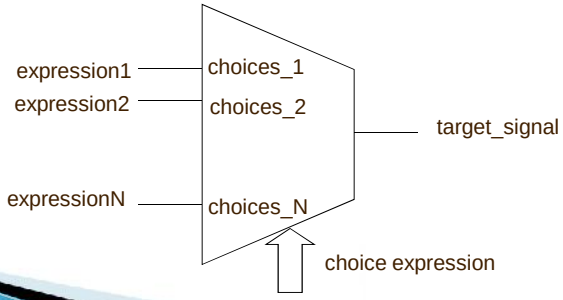
- **simple concurrent signal assignment**
  (Ĕ)
- **conditional concurrent signal assignment**
  (when-else)
- **selected concurrent signal assignment**
  (with-select-when)
- **generate scheme for equations**
  (for-generate)

27

Most Often Implied Structure

With –Select-When

```
with choice_expression select
   target_signal <= expression1 when choices_1,
                    expression2 when choices_2,
                        . . .
                    expressionN when choices_N;
```



28

Allowed formats of *choices_k*

WHEN value

WHEN value_1 | value_2 | .... | value N

WHEN OTHERS

```
WITH sel SELECT
    y <= a WHEN "000",
          c WHEN "001" | "111",
          d WHEN OTHERS;
```

29

Syntax

- Simplified syntax:

with select_expression **select**
  signal_name <=
    value_expr_1 **when** choice_1,
    value_expr_2 **when** choice_2,
    value_expr_3 **when** choice_3,
      . . .
    value_expr_n **when** choice_n;

30

Syntax

- select_expression
  - Discrete type or 1-D array
  - With finite possible values
- choice_i
  - A value of the data type
- Choices must be
  - mutually exclusive
  - all inclusive
  - **others** can be used as last choice_i

31

E.g., 4-to-1 mux

input s	output x
0 0	a
0 1	b
1 0	c
1 1	d

```
architecture sel_arch of mux4 is
begin
   with s select
     x <= a when "00",
          b when "01",
          c when "10",
          d when others;
end sel_arch ;
```

```
with s select
  x <= a when "00",
       b when "01",
       c when "10",
       d when "11";
```

32

E.g., 2-to-2^2 binary decoder

```
architecture sel_arch of decoder4 is
begin
   with sel select
     x <= "0001" when "00",
          "0010" when "01",
          "0100" when "10",
          "1000" when others;
end sel_arch ;
```

input s	output x
0 0	0001
0 1	0010
1 0	0100
1 1	1000

33

E.g., simple ALU

```
architecture sel_arch of simple_alu is
   signal sum, diff, inc: std_logic_vector(7 downto 0);
begin
   inc <= std_logic_vector(signed(src0)+1);
   sum <= std_logic_vector(signed(src0)+signed(src1));
   diff <= std_logic_vector(signed(src0)-signed(src1));
   with ctrl select
      result <= inc            when "000"|"001"|"010"|"011",
                sum            when "100",
                diff           when "101",
                src0 and src1  when "110",
                src0 or src1   when others;
end sel_arch;
```

input ctrl	output result
0--	src0 + 1
100	src0 + src1
101	src0 - src1
110	src0 and src1
111	src0 or src1

E.g., Truth table

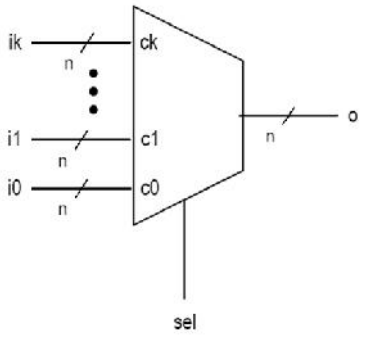
```
library ieee;
use ieee.std_logic_1164.all;
entity truth_table is
   port(
      a,b: in  std_logic;
      y: out   std_logic
   );
end truth_table;
architecture a of truth_table is
   signal tmp: std_logic_vector(1 downto 0);
begin
   tmp <= a & b;
   with tmp select
      y <= '0' when "00",
           '1' when "01",
           '1' when "10",
           '1' when others; -- "11"
end a;
```

input a b	output y
0 0	0
0 1	1
	1
	1

35

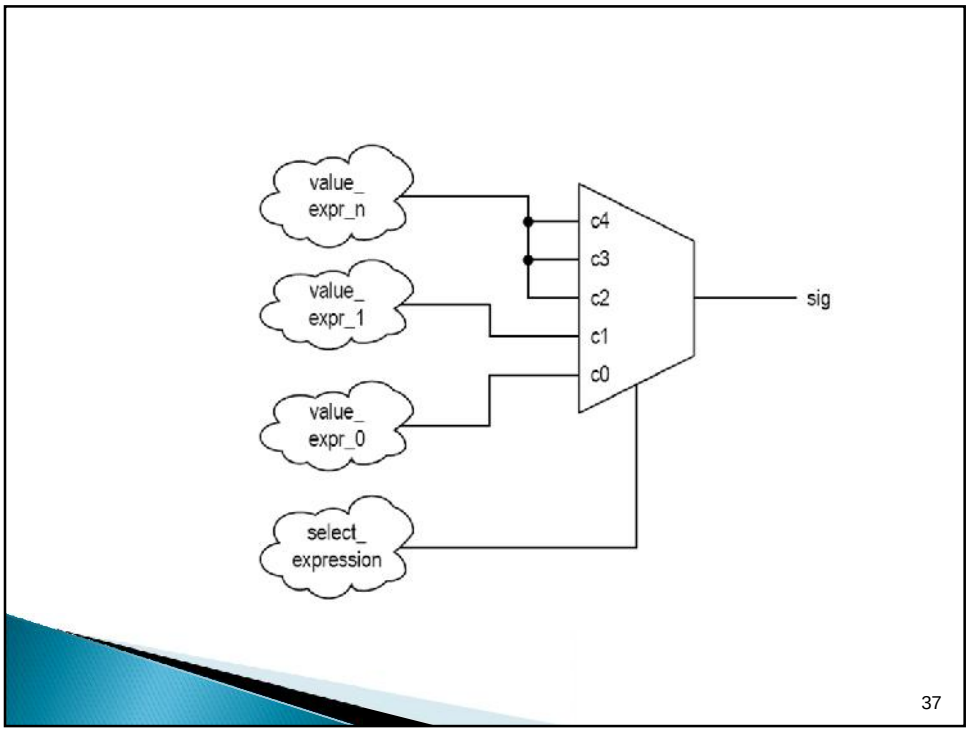
Conceptual implementation

- Achieved by a multiplexing circuit
- Abstract (k+1)-to-1 multiplexer
  - sel is with a data type of (k+1) values: c0, c1, c2, . . . , ck



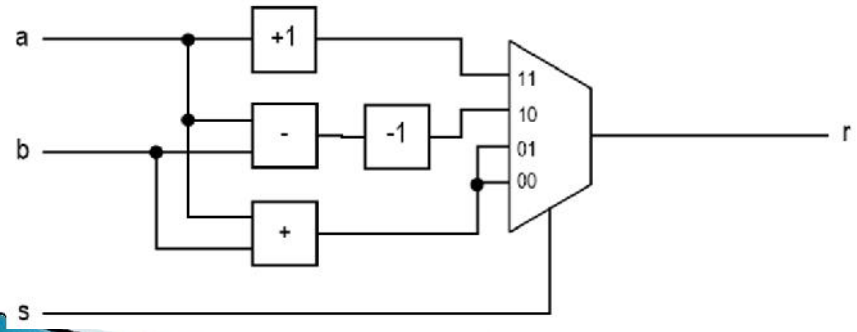

```
with select_expression select
   sig <= value_expr_0 when c0,
          value_expr_1 when c1,
          value_expr_n when others;
```

36



- E.g.,

```
signal a,b,r: unsigned(7 downto 0);
signal s: std_logic_vector(1 downto 0);
   . . .
   with s select
      r <= a+1    when "11",
           a-b-1  when "10",
           a+b    when others;
```



From selected assignment to conditional assignment

Conversion between conditional vs. selected signal assignment

```
with sel select
   sig <= value_expr_0 when c0,
          value_expr_1 when c1|c3|c5,
          value_expr_2 when c2|c4,
          value_expr_n when others;
```

```
sig <=
   value_expr_0 when (sel=c0) else
   value_expr_1 when (sel=c1) or (sel=c3) or (sel=c5) else
   value_expr_2 when (sel=c2) or (sel=c4) else
   value_expr_n;
```

39

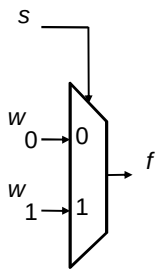
From conditional assignment to selected assignment

```
sig <= value_expr_0 when bool_exp_0 else
       value_expr_1 when bool_exp_1 else
       value_expr_2 when bool_exp_2 else
       value_expr_n;
```

```
sel(2) <= '1' when bool_exp_0 else '0';
sel(1) <= '1' when bool_exp_1 else '0';
sel(0) <= '1' when bool_exp_2 else '0';
with sel select
   sig <= value_expr_0 when "100"|"101"|"110"|"111",
          value_expr_1 when "010"|"011",
          value_expr_2 when "001",
          value_expr_n when others;
```

40

2-to-1 Multiplexer



(a) Graphical symbol

s	f
0	w_0
1	w_1

(b) Truth table

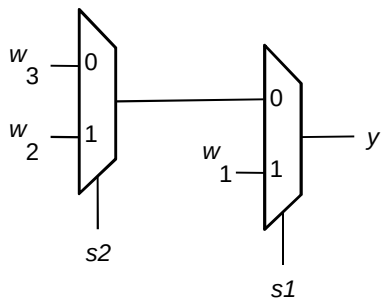
41

VHDL code for a 2-to-1 Multiplexer

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
    PORT ( w0, w1, s  : IN     STD_LOGIC ;
           f          : OUT    STD_LOGIC ) ;
END mux2to1 ;

ARCHITECTURE dataflow OF mux2to1 IS
BEGIN
    f <= w0 WHEN s = '0' ELSE w1 ;
END dataflow ;
```

42

Cascade of two multiplexers



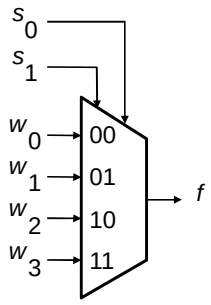
43

VHDL code for a cascade of two multiplexers

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux_cascade IS
    PORT (  w1, w2, w3: IN  STD_LOGIC ;
            s1, s2      : IN  STD_LOGIC ;
            f           : OUT   STD_LOGIC ) ;
END mux_cascade ;

ARCHITECTURE dataflow OF mux2to1 IS
BEGIN
    f <= w1 WHEN s1 = '1' ELSE
         w2  WHEN s2 = '1' ELSE
         w3 ;
END dataflow ;
```

44

4-to-1 Multiplexer



(a) Graphic symbol

s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

(b) Truth table

45

VHDL code for a 4-to-1 Multiplexer

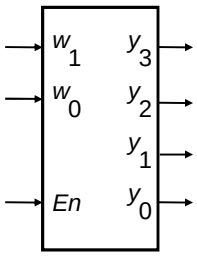
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux4to1 IS
    PORT (  w0, w1, w2, w3     : IN     STD_LOGIC ;
            s                  : IN     STD_LOGIC_VECTOR(1 DOWNTO 0) ;
            f                  : OUT    STD_LOGIC ) ;
END mux4to1 ;

ARCHITECTURE dataflow OF mux4to1 IS
BEGIN
    WITH s SELECT
        f <= w0 WHEN "00",
             w1 WHEN "01",
             w2 WHEN "10",
             w3 WHEN OTHERS ;
END dataflow ;
```

46

2-to-4 Decoder

En	w_1	w_0	y_3	y_2	y_1	y_0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0
0	x	x	0	0	0	0

(a) Truth table



(b) Graphical symbol

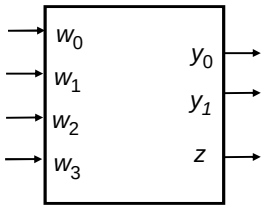
47

VHDL code for a 2-to-4 Decoder

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY dec2to4 IS
      PORT (   w    : IN      STD_LOGIC_VECTOR(1 DOWNTO 0) ;
               En   : IN      STD_LOGIC ;
               y    : OUT     STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END dec2to4 ;

ARCHITECTURE dataflow OF dec2to4 IS
      SIGNAL Enw : STD_LOGIC_VECTOR(2 DOWNTO 0) ;
BEGIN
      Enw <= En & w ;
      WITH Enw SELECT
               y <= "0001" WHEN "100",
                    "0010" WHEN "101",
                    "0100" WHEN "110",
                    "1000" WHEN "111",
                    "0000" WHEN OTHERS ;
END dataflow ;
```

48

Priority Encoder



w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

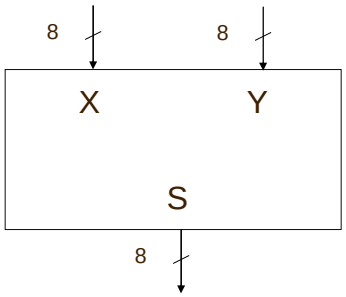
49

VHDL code for a Priority Encoder

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY priority IS
    PORT ( w   : IN     STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           y   : OUT    STD_LOGIC_VECTOR(1 DOWNTO 0) ;
           z   : OUT    STD_LOGIC ) ;
END priority ;

ARCHITECTURE dataflow OF priority IS
BEGIN
    y <=   "11" WHEN w(3) = '1' ELSE
           "10" WHEN w(2) = '1' ELSE
           "01" WHEN w(1) = '1' ELSE
           "00" ;
    z <= '0' WHEN w = "0000" ELSE '1' ;
END dataflow ;
```

50

Adder mod 2^8



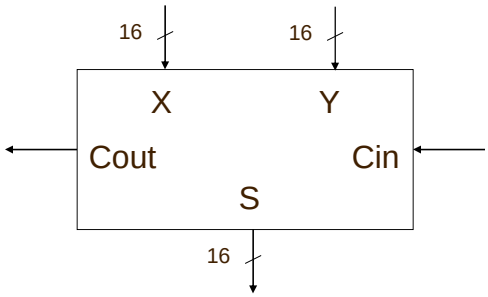
51

VHDL code for an Adder mod 2^8

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY adder16 IS
      PORT (   X                  : IN      STD_LOGIC_VECTOR(7 DOWNTO 0) ;
               Y                  : IN      STD_LOGIC_VECTOR(7 DOWNTO 0) ;
               S                  : OUT     STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
END adder16 ;

ARCHITECTURE dataflow OF adder16 IS
BEGIN
      S <= X + Y ;
END dataflow ;
```

52

16-bit Unsigned Adder



53

Operations on Unsigned Numbers

For operations on unsigned numbers

USE ieee.std_logic_unsigned.all
and
signals of the type
STD_LOGIC_VECTOR

OR

USE ieee.numeric_std.all,
signals of the type
UNSIGNED
and conversion functions:
std_logic_vector(), unsigned()

54

Signed and Unsigned Types

Behave exactly **like**
STD_LOGIC_VECTOR
plus, they determine whether a given vector should be treated as a signed or unsigned number.
Require
USE ieee.numeric_std.all;

55

VHDL code for a 16-bit Unsigned Adder

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY adder16 IS
    PORT (  Cin     : IN    STD_LOGIC ;
            X       : IN    STD_LOGIC_VECTOR(15 DOWNTO 0) ;
            Y       : IN    STD_LOGIC_VECTOR(15 DOWNTO 0) ;
            S       : OUT   STD_LOGIC_VECTOR(15 DOWNTO 0) ;
            Cout    : OUT   STD_LOGIC ) ;
END adder16 ;

ARCHITECTURE dataflow OF adder16 IS
    SIGNAL Sum : STD_LOGIC_VECTOR(16 DOWNTO 0) ;
BEGIN
    Sum <= ('0' & X) + Y + Cin ;
    S <= Sum(15 DOWNTO 0) ;
    Cout <= Sum(16) ;
END dataflow ;
```

56

Addition of Unsigned Numbers (1)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.numeric_std.all ;


ENTITY adder16 IS
      PORT ( Cin          : IN     STD_LOGIC ;
             X            : IN     STD_LOGIC_VECTOR(15 DOWNTO 0) ;
             Y            : IN     STD_LOGIC_VECTOR(15 DOWNTO 0) ;
             S            : OUT    STD_LOGIC_VECTOR(15 DOWNTO 0) ;
             Cout         : OUT    STD_LOGIC ) ;
END adder16 ;
```

57

Addition of Unsigned Numbers (2)

```
ARCHITECTURE dataflow OF adder16 IS
      SIGNAL Xu : UNSIGNED(15 DOWNTO 0);
      SIGNAL Yu: UNSIGNED(15 DOWNTO 0);
      SIGNAL Su : UNSIGNED(16 DOWNTO 0) ;
BEGIN
      Xu <= unsigned(X);
      Yu <= unsigned(Y);
      Su <= ('0' & Xu) + Yu + unsigned('0' & Cin) ;
      S <= std_logic_vector(Su(15 DOWNTO 0)) ;
      Cout <= Su(16) ;
END dataflow ;
```

58

Operations on Signed Numbers

For operations on signed numbers

USE ieee.std_logic_signed.all
and signals of the type
STD_LOGIC_VECTOR

OR

USE ieee.numeric_std.all,
signals of the type
SIGNED,
and conversion functions:
std_logic_vector(), signed()

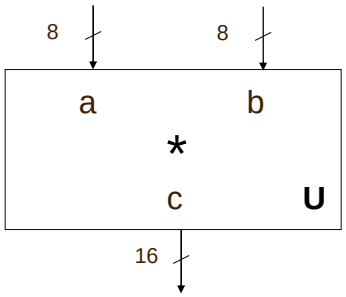
59

Unsigned vs. Signed Multiplication

Unsigned		Signed	
1111	15	1111	-1
x 1111	x 15	x 1111	x -1
11100001	225	00000001	1

60

8x8-bit Unsigned Multiplier



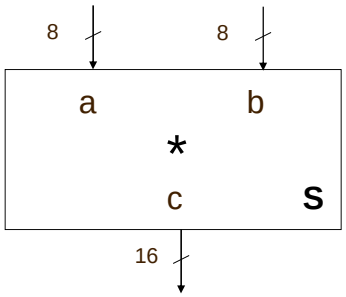
61

Multiplication of unsigned numbers

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all ;

entity multiply is
    port(
          a : in STD_LOGIC_VECTOR(7 downto 0);
          b : in STD_LOGIC_VECTOR(7 downto 0);
          c : out STD_LOGIC_VECTOR(15 downto 0)
       );
end multiply;

architecture dataflow of multiply is
begin
   c <= a * b;
end dataflow;
```

62

8x8-bit Signed Multiplier



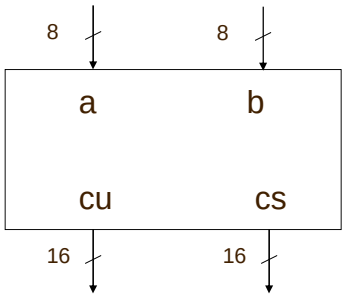
63

Multiplication of signed numbers

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all ;

entity multiply is
    port(
          a : in STD_LOGIC_VECTOR(7 downto 0);
          b : in STD_LOGIC_VECTOR(7 downto 0);
          c : out STD_LOGIC_VECTOR(15 downto 0)
       );
end multiply;

architecture dataflow of multiply is
begin
   c <= a * b;
end dataflow;
```

64

8x8-bit Unsigned and Signed Multiplier



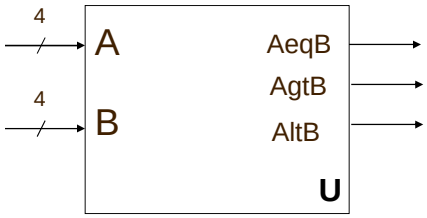
65

Multiplication of signed and unsigned numbers

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all ;

entity multiply is
    port(
            a : in STD_LOGIC_VECTOR(7 downto 0);
            b : in STD_LOGIC_VECTOR(7 downto 0);
            cu : out STD_LOGIC_VECTOR(15 downto 0);
            cs : out STD_LOGIC_VECTOR(15 downto 0)
        );
end multiply;

architecture dataflow of multiply is
begin

-- signed multiplication
   cs <= STD_LOGIC_VECTOR(SIGNED(a)*SIGNED(b));

-- unsigned multiplication
   cu <= STD_LOGIC_VECTOR(UNSIGNED(a)*UNSIGNED(b));
end dataflow;
```

66

4-bit Unsigned Number Comparator



67

VHDL code for a 4-bit Unsigned Number Comparator

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY compare IS
      PORT (   A, B                 : IN      STD_LOGIC_VECTOR(3 DOWNTO 0) ;
               AeqB, AgtB, AltB     : OUT     STD_LOGIC ) ;
END compare ;

ARCHITECTURE dataflow OF compare IS
BEGIN
      AeqB <= '1' WHEN A = B ELSE '0' ;
      AgtB <= '1' WHEN A > B ELSE '0' ;
      AltB <= '1' WHEN A < B ELSE '0' ;
END dataflow ;
```

68

VHDL code for a 4-bit Signed Number Comparator

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;

ENTITY compare IS
      PORT (   A, B                  : IN      STD_LOGIC_VECTOR(3 DOWNTO 0) ;
               AeqB, AgtB, AltB      : OUT     STD_LOGIC ) ;
END compare ;

ARCHITECTURE dataflow OF compare IS
BEGIN
      AeqB <= '1' WHEN  A = B ELSE '0' ;
      AgtB  <= '1' WHEN  A > B ELSE '0' ;
      AltB  <=  '1'  WHEN  A < B ELSE '0' ;
END dataflow ;
```

69

Arithmetic operations

Synthesizable arithmetic operations:

- Addition, +
- Subtraction, -
- Comparisons, >, >=, <, <=
- Multiplication, *
- Division by a power of 2, /2**6 (equivalent to right shift)

70

Arithmetic operations

The result of synthesis of an arithmetic operation is a
- combinational circuit
- without pipelining.

The exact internal **architecture** used (and thus delay and area of the circuit) **may depend** on the **timing constraints** specified during synthesis (e.g., the requested maximum clock frequency).

71

Integer Types

Operations on signals of the integer types:

INTEGER, NATURAL,

and their sybtypes, such as

TYPE day_of_month IS **RANGE 1 TO 31;**

are synthesizable in the range

$-(2^{31}-1)$ .. $2^{31}-1$ for INTEGERs and their subtypes

0 .. $2^{31}-1$ for NATURALs and their subtypes

72

Integer Types

Operations on signals (variables) of the integer types:

INTEGER, NATURAL,

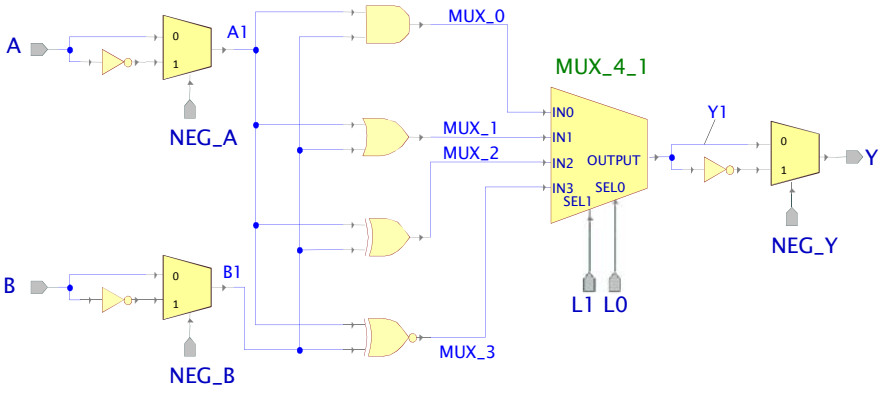
are less flexible and more difficult to control than operations on signals (variables) of the type

STD_LOGIC_VECTOR
UNSIGNED
SIGNED, and thus

are recommened to be avoided by beginners.

73

MLU Example
MLU Block Diagram



74

MLU: Entity Declaration

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mlu IS
   PORT(
        NEG_A : IN STD_LOGIC;
        NEG_B : IN STD_LOGIC;
        NEG_Y : IN STD_LOGIC;
        A :          IN STD_LOGIC;
        B :          IN STD_LOGIC;
        L1 :         IN STD_LOGIC;
        L0 :         IN STD_LOGIC;
        Y :          OUT STD_LOGIC
      );
END mlu;
```

75

MLU: Architecture Declarative Section

```
ARCHITECTURE mlu_dataflow OF mlu IS

SIGNAL  A1 :  STD_LOGIC;
SIGNAL  B1 :  STD_LOGIC;
SIGNAL  Y1 : STD_LOGIC;
SIGNAL  MUX_0 : STD_LOGIC;
SIGNAL  MUX_1 : STD_LOGIC;
SIGNAL  MUX_2 : STD_LOGIC;
SIGNAL  MUX_3 : STD_LOGIC;
SIGNAL  L: STD_LOGIC_VECTOR(1 DOWNTO 0);
```

76

MLU – Architecture Body

```
BEGIN
   A1 <= NOT A  WHEN (NEG_A='1') ELSE
         A;
   B1 <= NOT B  WHEN (NEG_B='1') ELSE
         B;
   Y <= NOT Y1 WHEN (NEG_Y='1') ELSE
         Y1;

   MUX_0 <= A1  AND  B1;
   MUX_1 <= A1   OR   B1;
   MUX_2 <= A1  XOR  B1;
   MUX_3 <= A1  XNOR B1;

    L <= L1 & L0;

   with (L) select
      Y1 <=  MUX_0  WHEN "00",
               MUX_1  WHEN "01",
             MUX_2  WHEN  "10",
              MUX_3  WHEN OTHERS;

END mlu_dataflow;
```

77

Combinational Logic Synthesis for Beginners

78

Simple rules for beginners

For combinational logic,
use only concurrent statements

- concurrent signal assignment (⇐)
- conditional concurrent signal assignment (when-else)
- selected concurrent signal assignment (with-select-when)
- generate scheme for equations (for-generate)

79

Simple rules for beginners

For circuits composed of
- simple logic operations (logic gates)
- simple arithmetic operations (addition, subtraction, multiplication)
- shifts/rotations by a constant

use

- concurrent signal assignment (⇐)

80

Simple rules for beginners

For circuits composed of
- multiplexers
- decoders, encoders
- tri-state buffers

use

- conditional concurrent signal assignment (when-else) (ending with **ELSE**)
- selected concurrent signal assignment (with-select-when) (ending with **WHEN OTHERS;**)

81

Example: VHDL code for a 4-to-1 MUX

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux4to1 IS
    PORT (  w0, w1, w2, w3    : IN     STD_LOGIC ;
            s                 : IN     STD_LOGIC_VECTOR(1 DOWNTO 0) ;
            f                 : OUT    STD_LOGIC ) ;
END mux4to1 ;

ARCHITECTURE dataflow OF mux4to1 IS
BEGIN
    WITH s SELECT
        f <= w0 WHEN "00",
             w1 WHEN "01",
             w2 WHEN "10",
             w3 WHEN OTHERS ;
END dataflow ;
```

82

when-else vs. with-select-when (1)

"when-else" should be used when:

1) there is only one condition (and thus, only one else), as in the 2-to-1 MUX

2) conditions are independent of each other (e.g., they test values of different signals)

3) conditions reflect priority (as in priority encoder); one with the highest priority need to be tested first.

83

when-else vs. with-select-when (2)

"with-select-when" should be used when there is

1) more than one condition

2) conditions are closely related to each other (e.g., represent different ranges of values of the same signal)

3) all conditions have the same priority (as in the 4-to-1 MUX).

84

Left vs. right side of the assignment

Left side <= **Right side**

<= when-else

with-select <=

- Internal signals (defined in a given architecture)
- Ports of the mode
  - **out**
  - inout

Expressions including:

- Internal signals (defined in a given architecture)
- Ports of the mode
  - **in**
  - inout

85